A-sharp: a Distributed A-star for Factored Planning

Loïg Jezequel, Eric Fabre

A#: a Distributed A* for Factored Planning

Loïg Jezequel, Eric Fabre

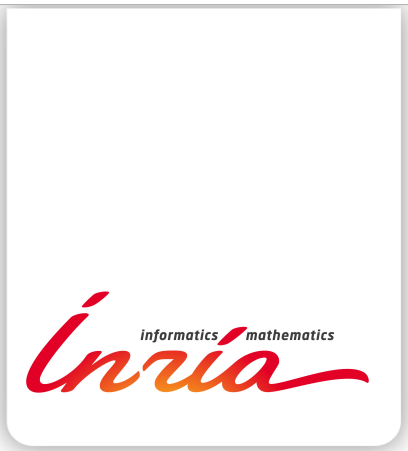
RESEARCH REPORT

N° 7927

March 2012

Project-Teams Distribcom

ISSN 0249-6399 ISRN INRIA/RR--7927--FR+ENG



A#: a Distributed A* for Factored Planning

Loïg Jezequel*, Eric Fabre†

Project-Teams Distribcom

Research Report n° 7927 — March 2012 — 17 pages

Abstract: Factored planning consists in driving a modular or distributed system to a target state, in an optimal manner, assuming all actions are controllable. Such problems take the form of path search in a product of graphs. The state space of each component is a graph, in which one must find a path to the local goal of this component. But when all components are considered jointly, the problem amounts to finding a path in each of these state graphs, while ensuring their compatibility on the actions that must be performed jointly by some components of the system. This paper proposes a solution under the form of a multi-agent version of A*. The proposed A# assembles several A*, each one performing a biased depth-first search in the graph of each component, and coordinates their exchanges to quickly find compatible paths to the goal.

Key-words: factored planning, distributed algorithms, paths search, optimization

* ENS Cachan Bretagne
† INRIA Rennes Bretagne Atlantique

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

A#: une version distribuée de A* pour la planification modulaire

Résumé : Faire de la planification modulaire c'est s'assurer qu'un système distribué atteigne (de manière optimale) un objectif en utilisant un ensemble d'actions, toutes considérées comme contrôlables. Résoudre ce type de problèmes revient à chercher un chemin dans un produit de graphes (chacun représentant l'un des composants du système considéré). Chercher un chemin local à chaque composant est relativement simple, cependant trouver de tels chemins qui soient compatibles entre eux (c'est à dire dont les actions partagées soient utilisées dans le même ordre) est bien plus délicat, notamment lorsque l'on ne souhaite pas calculer directement le produit de ces composants. Ce rapport de recherche propose une méthode pour résoudre de tels problèmes. Il s'agit en fait d'utiliser une version multi-agents de l'algorithme A^*, utilisant des informations venant des composants voisins pour biaiser la recherche de chemin de coût minimum au sein d'un graphe.

Mots-clés : planification modulaire, algorithmes distribués, recherche de chemins, optimisation

1 Introduction

Planning consists in organizing optimally a limited set of actions in order to reach some goal. Actions generally consume and produce resources, have a cost, and the goal is expressed as a desired value for some of these resources. From a control perspective, planning can also be regarded as driving an automaton to a target state, in an optimal manner, when all transitions are controllable. Each state then represents a tuple of values, one per resource, and transitions derive from the possible actions. In these terms, the problem amounts to finding a shortest path in a possibly huge weighted oriented graph, from an initial vertex to a set of possible final ones. Despite the NP-hardness of the problem, efficient algorithms have been proposed, as variants of the celebrated A* (pronounce 'A-star') [1]. The latter is nothing but a depth-first search, guided to the goal by some heuristic function, i.e. a lower-bound on the distance to the goal, available at each node. In practice, this approach performs much better than the worst case bound, that requires exploring the whole graph, provided heuristics are smartly designed [2, 3, 4].

Distributed planning addresses a similar problem, for a network of automata. This setting appears when resources are partitioned into N subsets, each one associated to the actions that modify them. Each such reduced planning problem can be represented as a smaller automaton (or component), or as a smaller graph, making it more tractable. But some actions simultaneously modify the resources of several of these N subsets, in other words some components may have to agree on some shared actions. One can thus view the problem as optimally driving a network of automata to a target state, each component having private and shared actions. Equivalently, this amounts to finding a path in a product graph. It was soon recognized that distributed or factored planning could render manageable some large scale planning problems that can not be addressed in a centralized manner.

Different approaches to distributed planning have been proposed [5, 6, 7], with the simpler objective to find a possible plan, not an optimal one. The idea being generally to find a local plan in some component, and take it as a seed or constraint to look for a compatible local plan in a neighbouring component, that is a component sharing actions with the first one. And so on, accross all components, with (possibly numerous) backtracks when impossibilities are encountered. Alternatively, [8] proposed an approach still based on a message passing strategy, but handling weighted automata to perfom computations. This solution actually provides all possible (distributed) plans, and identifies the best one(s). The present paper adopts another strategy and aims at a true distributed version of A*. The idea is to run modified versions of A* in parallel, one per component, and to bias the search of each one in order to favor the exploration of local paths/plans that are likely to be compatible with the ones explored in neighbouring components. Each such local A* thus has to inform its neighbors of the shared actions that are likely to lead to a solution, from its perspective. Such communicating parallel versions of A* suggested the name A# (pronounce 'A-sharp').

The paper is organized as follows. The optimal distributed planning is first formalized and illustrated on a running example (Section II). Then a simplified version of the problem is examined, in order to clarify the mechanism of the proposed approach in the simple case of two components (Section III). These principles are then generalized, and it is proved that the algorithm converges and yields the desired result (Section IV).

2 Planning for Distributed Systems

There exist several manners to set up a planning problem. For simplicity, this paper presents planning as an optimal path search problem in a graph.

2.1 Planning as path search in a graph

Let $\mathcal{G} = (V, E)$ be a (finite) directed graph, with V as set of vertices and $E \subseteq V \times V$ as set of edges. A *path* in $\mathcal{G}$ is a sequence of edges $p = e_1 \dots e_n$ such that, for any $1 \leq i < n$, one has $e_i = (v_i, v_{i+1})$. p is said to be a path from $v_1 = p^-$ to $v_{n+1} = p^+$. A labelling of $\mathcal{G}$ is a function $\lambda : E \to \Lambda$, where Λ is a finite set of labels, also called *actions* in the sequel. The labelling extends to paths $p = e_1 \dots e_n$ by $\lambda(p) = \lambda(e_1) \dots \lambda(e_n) \in \Lambda^*$. This labeling is *deterministic* iff for every pair of edges (v, v') and (v, v''), $\lambda(v, v') = a = \lambda(v, v'')$ entails $v' = v''$. In other words, the effect of action a at vertex v has a unique outcome. For simplicity, and to match standard planning problems, this paper considers deterministic labelings. By abuse of notation, we sometimes do not distinguish p and $\lambda(p)$. Similarly to labels, a cost function on $\mathcal{G}$ is defined as $c : E \to \mathbb{R}_+$, and associates costs to edges. It also extends to paths by $c(p) = \sum_{i=1}^{n} c(e_i)$.

A *planning problem* is now defined as a decorated graph $\mathcal{P} = (V, E, \Lambda, \lambda, c, i, F)$ where $i \in V$ is an initial vertex, and $F \subseteq V$ is a set of possible final (or goal) vertices. The objective is to find a path p such that $p^- = i,\ p^+ \in F$ and $c(p)$ is minimal among such paths. The word $\lambda(p)$ is called the (action) plan.

2.2 The A* solution

A* is a depth-first exploration strategy in $\mathcal{P}$ [1]. Therefore it is based on a set of active or open vertices $O \subseteq V$, initialized to $O = \{i\}$, that forms the (inner) boundary of the explored region $S \subseteq V$, progressively expanded to reach F. To each vertex v in O one associates two values: $g(v)$ and $h(v)$. $g(v)$ is the cost of the shortest path from i to v within the subgraph $\mathcal{P}_{|S}$ ($\mathcal{P}$ restricted to S). $h(v)$ is a *heuristic function*, that is a lower bound on the cost to reach F from v in $\mathcal{P}$. So $h(v) = 0$ for $v \in F$. Function h is often required to be consistent, i.e. to satisfy $h(v) \leq c(v, v') + h(v')$ for any edge $(v, v') \in E$. The heuristic h is used to guide the search towards the target F. Its selection depends very much on the nature of the planning problem, and influences greatly the performance of A* [2, 3, 4].

The algorithm proceeds as follows: it recursively 'expands' the most promising vertex in O, i.e. the vertex $v \in O$ with ranking $r(v) \triangleq g(v) + h(v) \leq r(v')$ for any other $v' \in O$. Expansion means that every successor v' of v in $\mathcal{P}$ is examined. Its g value is updated according to $g^{new}(v') = min(g^{old}(v'), g(v) + c(v, v'))$ (with $g^{old}(v') = +\infty$ when $v' \notin S$, i.e. when v' has not been visited),. If $g^{new}(v') < g^{old}(v')$, then v' is (re)activated, i.e. (re)placed into O. The expanded vertex v is then removed from O, but it remains in the set of visited nodes S. The algorithm stops when the vertex v chosen for expansion already belongs to F. The best path p from i to v yields an optimal action plan. Path p can be easily recovered by backtracking from $p^+ \in F$, provided at every expansion step, one stores the predecessor v on the best path to v'.

2.3 Distributed planning

Let $\mathcal{P}_1, \mathcal{P}_2$ be two planning problems, with $\mathcal{P}_k = (V_k, E_k, \Lambda_k, \lambda_k, c_k, i_k, F_k)$ for $k = 1, 2$, and such that $\Lambda_1 \cap \Lambda_2 \neq \emptyset$. We define a distributed (or factored) optimal planning problem as a pair[1] $(\mathcal{P}_1, \mathcal{P}_2)$ of such interacting planning problems. The actions in $\Lambda_1 \cap \Lambda_2$ are said to be common or synchronized actions between the two problems, while those in $\Lambda_1 \setminus \Lambda_2$ (resp. $\Lambda_2 \setminus \Lambda_1$) are private to $\mathcal{P}_1$ (resp. $\mathcal{P}_2$). This setting is extremely natural in practical planning problems: $\mathcal{P}_1$ and $\mathcal{P}_2$ can represent the state of disjoints subsets of resources, that can be modified jointly by some actions. As an illustration, imagine $\mathcal{P}_1$ as the problem of transporting parts from different

[1] We limit ourselves to two components for a matter of simplicity, because this is sufficient to expose the concepts of our approach. But the results of this paper extend to N components, which will be detailed in forthcoming publications.

warehouses to an assembly plant, and $\mathcal{P}_2$ as the problem of optimizing the route of the truck in charge of this transport. Interactions occur only at loading and unloading.

A distributed planning problem $(\mathcal{P}_1, \mathcal{P}_2)$ can be recast into a standard planning problem $\mathcal{P}$ by means of a product operation: $\mathcal{P} = \mathcal{P}_1 \times \mathcal{P}_2$ (similar to the synchronous product of labeled automata). This operation is defined as $V = V_1 \times V_2$, $i = (i_1, i_2)$, $F = F_1 \times F_2$, $\Lambda = \Lambda_1 \cup \Lambda_2$. For the edges, one has $E = E_s \uplus E_{p,1} \uplus E_{p,2}$ where E_s denotes synchronized transitions and $E_{p,k}$ the private transitions of $\mathcal{P}_k$. They are given by $E_s = \{((v_1, v_2), (v_1', v_2')) : (v_k, v_k') \in E_k,\ \lambda_1(v_1, v_1') = \lambda_2(v_2, v_2')\}$, while private moves of $\mathcal{P}_1$ assume $\mathcal{P}_2$ remains idle $E_{p,1} = \{(v_1, v_2), (v_1', v_2)) : (v_1, v_1') \in E_1,\ v_2 \in V_2,\ \lambda_1(v_1, v_1') \notin \Lambda_2\}$ and symmetrically for $E_{p,2}$. Labels follow accordingly: $\lambda((v_1, v_2), (v_1', v_2')) = \lambda_1(v_1, v_1') = \lambda_2(v_2, v_2')$ for E_s, and $\lambda((v_1, v_2), (v_1', v_2)) = \lambda_1(v_1, v_1')$ for $E_{p,1}$ (sym. for $E_{p,2}$). In the same way, costs are additive: $c((v_1, v_2), (v_1', v_2')) = c_1(v_1, v_1') + c_2(v_2, v_2')$ for E_s, and $c((v_1, v_2), (v_1', v_2)) = c_1(v_1, v_1')$ for $E_{p,1}$ (sym. for $E_{p,2}$).

The resulting product problem $\mathcal{P} = \mathcal{P}_1 \times \mathcal{P}_2$ may however be very large compared to the $\mathcal{P}_k$ alone (see Figure 1). This is the usual state explosion problem, which is doubled by the emergence of so-called concurrency diamonds on edges, i.e. the interleaving of private actions of $\mathcal{P}_1$ and $\mathcal{P}_2$.

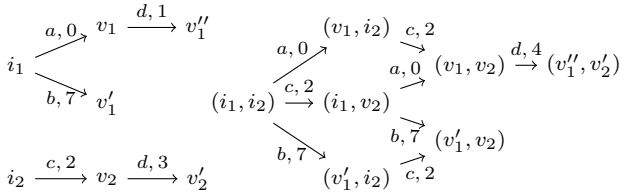


Figure 1: Two planning problems (left) and their product (right).

Distributed planning aims at avoiding this double explosion of problem $\mathcal{P}$. The idea is to look for a *pair* of paths (p_1, p_2), such that

1. p_k defines a valid plan in $\mathcal{P}_k$, not necessarily optimal, for $k = 1, 2$
2. the two paths p_1 and p_2 are *compatible*, i.e. they coincinde on shared actions, which we translate by $\pi_{\Lambda_2}(\lambda_1(p_1)) = \pi_{\Lambda_1}(\lambda_2(p_2))$ (see below),
3. the pair (p_1, p_2) is jointly optimal, i.e. $c(p_1) + c_2(p_2)$ is minimal among pairs of paths satisfying the two above criteria.

The natural projection $\pi_{\Lambda'}(w)$ of a word $w \in \Lambda^*$ on the subalphabet $\Lambda' \subseteq \Lambda$ is defined by $\pi_{\Lambda'}(\epsilon) = \epsilon$ for the empty word, and by $\pi_{\Lambda'}(aw) = a\pi_{\Lambda'}(w)$ if $a \in \Lambda'$, and $\pi_{\Lambda'}(aw) = \pi_{\Lambda'}(w)$ otherwise. The compatibility of p_1 and p_2 thus induces that there exists a global path p for problem $\mathcal{P}$ such that $\pi_{\Lambda_k}(\lambda(p)) = \lambda_k(p_k)$, $k = 1, 2$. Such a path p corresponds to an interleaving of the actions in p_1 and p_2. It is generally not unique: for example in Fig. 1 the distributed plan (ad, cd) corresponds to two global plans *acd* and *cad* where private (concurrent) actions are interleaved in different manners. This reveals that distributed planning actually aims at building *partially ordered plans*, which is a powerful way to turn the explosion due to concurrency into an advantage: one saves the exploration of meaningless interleavings.

2.4 Coordinated parallel path searches

The approach proposed in this paper consists in associating an agent φ_k to each problem $\mathcal{P}_k$. Each agent performs an A*-like search in its local graph, and takes into account the constraints and

costs of the other agent through an appropriate communication mechanism. Communications are asynchronous and can take place at any time. Nevertheless, it is proved that the algorithm converges to a distributed optimal plan (p_1, p_2).

For a matter of clarity, the next section first addresses a simpler problem called *compatible final states (CFS)*. $\mathcal{P}_1$ and $\mathcal{P}_2$ have no common actions, so $\Lambda_1 \cap \Lambda_2 = \emptyset$. However, their final states are 'colored' by functions $\gamma_k : F_k \to \Gamma$ where Γ is a finite color set. The CFS problem amounts to finding an optimal distributed plan (p_1, p_2) where the compatibility condition (2) above is replaced by a compatibility of their final states: $\gamma_1(p_1^+) = \gamma_2(p_2^+)$. We shall assume that there is a unique optimal (common) final color, if ever the CFS problem has a solution. Otherwise selecting one optimal final color among several becomes an agreement problem, that can be solved on top of our approach.

The standard distributed planning problem can be reduced to the CFS as follows. First $\Lambda_1 \cap \Lambda_2 \neq \emptyset$ is ignored. Then, instead of assuming that the colors of final states are given beforehand, one *computes* them as functions of the current explored paths to reach any vertex of graph $\mathcal{P}_k$. Specifically, a path p_k in $\mathcal{P}_k$ with $p_k^+ \in F_k$ will have $\pi_{\Lambda_1 \cap \Lambda_2}(\lambda_k(p_k))$ as final 'color.' Compatibility of final colors thus entails the compatibility of p_1, p_2 in terms of shared actions. The main difference with CFS is thus that the color set becomes infinite: $\Gamma = (\Lambda_1 \cap \Lambda_2)^*$.

3 Compatible final states

3.1 Intuition on the approach

Let $\{k, \bar{k}\} = \{1, 2\}$. The agent φ_k attached to problem $\mathcal{P}_k$ relies on four functions. Two relate to the standard shape of a local A*: $g_k : V_k \to \mathbb{R}$ yields the (current) best known cost to reach any $v \in V_k$ and $h_k : V_k \times \Gamma \to \mathbb{R}$ is a set of heuristic functions towards F_k, one per terminal color. Equivalently, one has a heuristic function towards any $F_k \cap \gamma^{-1}(c)$ for $c \in \Gamma$. Besides, two other functions inform φ_k on the state of the search in $\mathcal{P}_{\bar{k}}$, where $\{k, \bar{k}\} = \{1, 2\}$. Namely, one has $H_{\bar{k}} : \Gamma \to \mathbb{R}$, and $G_{\bar{k}} : \Gamma \to \mathbb{R}$. $H_{\bar{k}}$ is a (generally) time varying heuristic that measures how much color $c \in \Gamma$ is promising at the current point of resolution of problem $\mathcal{P}_{\bar{k}}$. Similarly, $G_{\bar{k}}(c)$ indicates the current best cost for reaching color c in $\mathcal{P}_{\bar{k}}$. Both values are asynchronously updated by agent $\varphi_{\bar{k}}$ under the form of messages sent to φ_k.

We now formalize these features and explain how these four functions are used and updated by each agent, how termination is detected by both of them, and how an optimal distributed plan solving the CFS is extracted.

3.2 Proposed algorithm

Let us consider first a non-varying distant heuristic $H_{\bar{k}}$: $H_{\bar{k}}(c) = h_{\bar{k}}(i_{\bar{k}}, c)$ for all $c \in \Gamma$. To the distant cost function on final colors $G_{\bar{k}}$ one associates an oracle $\Theta_{\bar{k}} : \Gamma \to \{null, optimal, useless\}$, with the following meaning. $\Theta_{\bar{k}}(c) = optimal$ means that a best plan towards final vertices of color c is known in $\mathcal{P}_{\bar{k}}$, and in that case $G_{\bar{k}}(c)$ represents the optimal cost to reach color c in $\mathcal{P}_{\bar{k}}$. $\Theta_{\bar{k}}(c) = useless$ means that $\varphi_{\bar{k}}$ can guarantee that for sure no optimal distributed plan $(p_k, p_{\bar{k}})$ exists which terminates in color c, and *null* is the remaining default (and initial) value of $\Theta_{\bar{k}}$. This oracle satisfies the following property: for every color $c \in \Gamma$, there exists a finite time at which $\Theta_{\bar{k}}(c)$ jumps from *null* to either *optimal* or *useless*, and keeps this value forever.

Each agent φ_k executes the variant of A* given in Algorithm 1. Vertices can be marked in three different ways: open, closed, or candidate. A candidate vertex v belongs to F_k, and thus represents a local plan in $\mathcal{P}_k$ that can be proposed to $\varphi_{\bar{k}}$ as a possible local component of a distributed plan. Initially all vertices v in $V_k \setminus \{i_k\}$ are closed and satisfy $g_k(v) = +\infty$. To

progressively open them and explore graph $\mathcal{P}_k$, one relies on the ranking function R_k defined as follows. If $v \in V_k$ is not candidate

$$R_k(v) \quad = \quad g_k(v) + \min_{c \in \Gamma} (h_k(v,c) + H_{\bar{k}}(c))$$

which integrates the cost of color c for agent $\varphi_{\bar{k}}$, and then optimizes on the possible final color. For a candidate vertex v, one takes

$$\begin{aligned} R_k(v) &= g_k(v) + G_{\bar{k}}(\gamma_k(v)) \text{ if } \Theta_{\bar{k}}(\gamma_k(v)) = \mathit{optimal} \\ &= g_k(v) + H_{\bar{k}}(\gamma_k(v)) \text{ otherwise} \end{aligned}$$

which associates to the possible final vertex v the cost of its color $\gamma_k(v)$ for agent $\varphi_{\bar{k}}$.

The recursive (local) search then proceeds as follows. At each iteration φ_k selects the most promising non-closed (i.e. open or candidate) vertex v, i.e. the one that minimizes the ranking function R_k. According to the nature of v, agent φ_k either a) progresses in the exploration of $\mathcal{P}_k$ using an expansion function (Algorithm 2), this is the case in particular when v is open, or b) checks whether it can draw some conclusion using the information provided by the other agent $\varphi_{\bar{k}}$. These conclusions can be (1) that v is the goal vertex reached by a path part of a globally optimal plan (line 8), (2) that v will never be the goal vertex reached by a path part of a globally optimal plan (line 13), or (3) nothing for the moment (line 15). The reader familiar with A* may thus immediately identify its shape within Algorithm 1. The main difference lies in the stopping condition, due to the necessity to take into account constraints transmitted by the other agent.

Algorithm 1 executed by φ_k

```
 1: mark i_k open; g_k(i_k) ← 0; calculate R_k(i_k)
 2: while there exists non-closed vertices do
 3:    let v be the non-closed vertex with minimal R_k(v)
 4:    if v is open then
 5:       expand(v)
 6:    else
 7:       case: Θ_k̄(γ_k(v)) = optimal
 8:          if R_k(v) = g_k(v) + G_k̄(γ_k(v)) then
 9:             return v and terminate
10:          else
11:             calculate R_k(v)
12:          end if
13:       case: Θ_k̄(γ_k(v)) = useless
14:          mark v closed
15:       case: Θ_k̄(γ_k(v)) = null
16:          if there exists open vertices then
17:             let v' be the open vertex with minimal R_k(v')
18:             expand(v')
19:          end if
20:    end if
21: end while
```

Notice that the call to the *expand* function at line 18 of Algorithm 1 is not required for termination nor validity, however it will allow agent $\varphi_{\bar{k}}$ to maintain $G_{\bar{k}}$ and $\Theta_{\bar{k}}$ using its own instance of Algorithm 1. Otherwise $\varphi_{\bar{k}}$ should run a standard A^* algorithm in parallel with Algorithm 1.

Algorithm 2 expand function

```
1: if v ∈ F_k then
2:    mark v candidate
3:    calculate R_k(v)
4: else
5:    mark v closed
6: end if
7: for all v' such that (v, v') ∈ E_k do
8:    g_k(v') ← min(g_k(v'), g_k(v) + c_k((v, v')))
9:    if g_k(v') strictly decreased then
10:      mark v' open
11:      pred(v') ← v
12:   end if
13:   calculate R_k(v')
14: end for
```

Theorem 1. *In this context, any execution of Algorithm 1 by φ_k on $\mathcal{P}_k$ terminates. Moreover, if the CFS problem $(\mathcal{P}_1, \mathcal{P}_2)$ has a solution, the output of Algorithm 1 for agent φ_k is a goal vertex $v_k \in F_k$, reached by a local plan p_k. The assembling of p_1 and p_2 provided by agents φ_1 and φ_2 resp. yields an optimal distributed plan (p_1, p_2) solving $(\mathcal{P}_1, \mathcal{P}_2)$.*

Theorem 1 is proved in three steps: first termination is proved (Proposition 1), then existence of an output when a distributed plan exists is proved (Proposition 2), and finally the fact that the output provides an optimal distributed plan is proved (Proposition 3).

Proposition 1. *Algorithm 1 terminates when executed by φ_k on $\mathcal{P}_k$.*

Proof. Suppose φ_k executes Algorithm 1 on $\mathcal{P}_k$ without terminating. It means that there always exists a vertex which is either open or candidate (else the while loop would stop). It also means that the vertex with the smallest R_k value never fulfills the condition of line 8.

Moreover it is not possible to satisfy the condition of line 13 an infinite number of times in a row. This is because (1) V_k is finite, and thus there exists a finite number of possible candidate, and (2) when the condition of line 13 is satisfied a candidate becomes closed an no new vertex becomes candidate.

This implies that the *expand* function will be called at finite time intervals while there exists open vertices. Hence, after some time all vertices will be either closed or candidate. This is due to the facts that (1) each call to *expand* makes an open vertex become closed or candidate, (2) V_k is finite, and thus there exists a finite number of possible open vertices, (3) each v marked as open by *expand* is such that $g_k(v)$ strictly decreased (line 9 of *expand*) and even, from the structure of the problems considered, one has that $g_k(v)$ strictly decreased of at least some constant c which is the minimal non zero difference between the costs of any two transitions from E_k, and (4) for a given v it is not possible that $g_k(v) < 0$ (this is due to the initialization of $g_k(i_k)$, line 1).

As soon as all vertices are either closed or candidate, no new vertex can become open. This is because only *expand* function can open vertices and the conditions to call *expand* require an open vertex (lines 16 and 4). Moreover, no new vertices will become candidate, for the same reason.

By definition of $\Theta_{\bar{k}}$, for any color $c \in \Gamma$ there exists a time after which either $\Theta_{\bar{k}}(c) = \mathit{optimal}$ or $\Theta_{\bar{k}}(c) = \mathit{useless}$. In particular, for any candidate (that is for any non-closed) vertex v, after some time, $\Theta_{\bar{k}}(\gamma_k(v)) = \mathit{optimal}$ or $\Theta_{\bar{k}}(\gamma_k(v)) = \mathit{useless}$. Consider the time after which, for

any candidate vertex v, either $\Theta_{\bar{k}}(\gamma_k(v)) = \mathit{optimal}$ or $\Theta_{\bar{k}}(\gamma_k(v)) = \mathit{useless}$. Consider the vertex v selected. Two cases are possible: (1) $\Theta_{\bar{k}}(\gamma_k(v)) = \mathit{useless}$, v is closed, the number of candidate vertices strictly decreases, (2) $\Theta_{\bar{k}}(\gamma_k(v)) = \mathit{optimal}$, either algorithm terminate or $R_k(v)$ is calculated and becomes equal to $g_k(v) + G_{\bar{k}}(\gamma_k(v))$ (and thus if v is selected later the algorithm will terminate). This proves that, after some time, either all vertices will be closed (if $\Theta_{\bar{k}}(\gamma_k(v)) = \mathit{useless}$ for all v candidate) or condition of line 8 will be satisfied. Both these cases are in contradiction with the hypothesis taken that φ_k executes Algorithm 1 on $\mathcal{P}_k$ without terminating. This proves Proposition 1. □

Proposition 2. *Algorithm 1 outputs some v when executed by φ_k on $\mathcal{P}_k$ if and only if there exists a solution.*

Proof. Suppose Algorithm 1 outputs no v when executed by φ_k on $\mathcal{P}_k$. It means it terminated because all vertices has been marked closed (Proposition 1). Thus all reachable goal vertices have been marked as candidate, by definition of *expand* function. Then all candidate vertices have been marked as closed, meaning that for any v candidate $\Theta_{\bar{k}}(\gamma_k(v)) = \mathit{useless}$. If there exists a solution it means that there exists some $v \in F_k$, reachable in $\mathcal{P}_k$, such that $\Theta_{\bar{k}}(\gamma_k(v)) = \mathit{optimal}$ at some time. This is not compatible with the facts that all reachable goal vertices have been marked as candidate and that for any v candidate $\Theta_{\bar{k}}(\gamma_k(v)) = \mathit{useless}$. Hence, if Algorithm 1 outputs no v when executed by φ_k on $\mathcal{P}_k$ then there exists no solution. Which proves that if there exists a solution then Algorithm 1 outputs some v when executed by φ_k on $\mathcal{P}_k$.

Suppose Algorithm 1 outputs some v when executed by φ_k on $\mathcal{P}_k$. It means that v has been marked candidate at some point. Which, by construction implies that $v \in F_k$ and there exists a path from i_k to v in $\mathcal{P}_k$. It also means that $\Theta_{\bar{k}}(\gamma_k(v)) = \mathit{optimal}$, which, by definition of $\Theta_{\bar{k}}$ implies that there exists a path in $\mathcal{P}_{\bar{k}}$ from $i_{\bar{k}}$ to some goal vertex with color $\gamma_k(v)$. Thus, there is a path in $\mathcal{P}_k$ reaching a goal vertex of color $\gamma_k(v)$ and there is a path in $\mathcal{G}_{\bar{k}}$ reaching a goal vertex of color $\gamma_k(v)$. This exactly means that there exists a solution. Which proves that if Algorithm 1 outputs some v when executed by φ_k on $\mathcal{P}_k$ then there exists a solution. □

Proposition 3. *When Algorithm 1 outputs some v, when executed by φ_k on $\mathcal{P}_k$, it is the goal vertex reached by a path p_k such that there is an optimal distributed plan $(p_k, p_{\bar{k}})$.*

Proof. Notice that any output v of Algorithm 1 is necessarily a goal vertex, by construction. Suppose Algorithm 1 outputs v, a goal vertex reached by a path p_k such that for any $p_{\bar{k}}$, $(p_k, p_{\bar{k}})$ is not an optimal distributed plan. Denote by v' a goal vertex reached by a path p'_k such that there exists an optimal distributed plan $(p'_k, p'_{\bar{k}})$ and $\Theta_{\bar{k}}(\gamma_k(v')) = \mathit{optimal}$ after some time. Such a v' exists because, as Algorithm 1 outputs v, by Proposition 2 there exists a solution to the considered planning problem. When Algorithm 1 stops by outputting v, two cases are possible: either (1) $g_k(v')$ is the optimal cost for reaching v' or (2) it is strictly greater than this optimal cost.

Consider case (1). For sure, v' is either open or candidate. The fact that $g_k(v') < \infty$ implies that v' has been marked as open at some time. From that, as $v' \in F_k$, it is not possible that v' has been marked as closed at line 5 of *expand* function. Moreover, as $\Theta_{\bar{k}}(\gamma_k(v')) = \mathit{optimal}$ after some time, it is not possible that $\Theta_{\bar{k}}(\gamma_k(v'))$ has been equal to $\mathit{useless}$, and so it is not possible that v' has been marked as closed at line 14 of Algorithm 1. As $g_k(v')$ is the optimal cost for reaching v', one has $g_k(v') < g_k(v)$ because v is not part of a globally optimal plan. Moreover $H_{\bar{k}}(\gamma_k(v')) \leq G_{\bar{k}}(\gamma_k(v))$ by definition and if $\Theta_{\bar{k}}(\gamma_k(v)) = \mathit{optimal}$, $G_{\bar{k}}(\gamma_k(v')) \leq G_{\bar{k}}(\gamma_k(v))$. In conclusion one has $R_k(v') < R_k(v)$. This implies that, at line 3 of Algorithm 1, it is never possible to select v before v'. And, for this reason it is not possible to output v.

Consider case (2). Using the same argument than in the proof of the original A^* algorithm, one can show that there exists an open vertex v'' such that: v'' is on a p'_k and $g_k(v'')$ is optimal. For the same reason as for case (1) it is not possible to select v before v'' at line 3 of Algorithm 1. And, so it is not possible to output v.

In both cases a contradiction has been given with the fact that Algorithm 1 can output a goal vertex reached by a path reached by a path p_k such that for any $p_{\bar{k}}$, $(p_k, p_{\bar{k}})$ is not an optimal distributed plan. This proves that when Algorithm 1 outputs some v, it is the goal vertex reached by a path p_k such that there is an optimal distributed plan $(p_k, p_{\bar{k}})$. □

3.3 Implementation of $G_{\bar{k}}$

The remaining of this section gives a feasible construction of the distant (color) cost function $G_{\bar{k}}$ and of the oracle $\Theta_{\bar{k}}$, showing that Algorithm 1 is usable in practice. These two functions have to be computed by agent $\varphi_{\bar{k}}$ independently of problem $\mathcal{P}_k$, and in particular, independently of G_k and Θ_k. The *expand* function is considered atomic: no update of $\Theta_{\bar{k}}$ or $G_{\bar{k}}$ will occur during the execution of this function by φ_k.

A possible implementation follows, where $\Theta_{\bar{k}}$ and $G_{\bar{k}}$ are computed within Algorithm 1 by $\varphi_{\bar{k}}$:

- initialization: $\forall c \in \Gamma,\ G_{\bar{k}}(c) = +\infty,$ and if $F_{\bar{k}} \cap \gamma_{\bar{k}}^{-1}(c) = \emptyset$ then $\Theta_{\bar{k}}(c) = \mathit{useless}$ otherwise $\Theta_{\bar{k}}(c) = \mathit{null}$,
- update: as soon as some final vertex $v \in F_{\bar{k}}$ is open or candidate, if no other open vertex $v' \in V_{\bar{k}}$ satisfies $g_{\bar{k}}(v') + h_{\bar{k}}(v', \gamma_{\bar{k}}(v)) < g_{\bar{k}}(v)$, then color $\gamma_{\bar{k}}(v)$ can not be reached with a lower cost in $\mathcal{P}_{\bar{k}}$, so $\Theta_{\bar{k}}(\gamma_{\bar{k}}(v))$ is set to *optimal* and

$$G_{\bar{k}}(\gamma_{\bar{k}}(v)) = \min_{v' \in F_{\bar{k}}, \gamma_{\bar{k}}(v') = \gamma_{\bar{k}}(v)} g_{\bar{k}}(v')$$

- final update: when Algorithm 1 stops, for all $c \in \Gamma$ such that $\Theta_{\bar{k}}(c) = \mathit{null}$, set $\Theta_{\bar{k}}(c) = \mathit{useless}$, and $G_{\bar{k}}(c) = +\infty$.

Proposition 4. *For any $c \in \Gamma$ there exists a finite time after which either $\Theta_{\bar{k}}(c) = \mathit{optimal}$ or $\Theta_{\bar{k}}(c) = \mathit{useless}$. Moreover, as soon as the value of $\Theta_{\bar{k}}(c)$ is different from null it no longer changes.*

Proof. Recall that all vertices are accessible in $\mathcal{P}_{\bar{k}}$. One can not rely on the values of Θ_k to prove Proposition 4, so, for that purpose, Proposition 1 can not be considered as true, the case where Algorithm 1 does not terminates thus has to be considered. Consider $c \in \Gamma$. If no goal vertex with color c exists, then, $\Theta_{\bar{k}}(c) = \mathit{useless}$ from the beginning. Else, two cases are possible: (1) Algorithm 1 terminates, and (2) Algorithm 1 does not terminate.

In case (1) all $c \in \Gamma$ for which $\Theta_{\bar{k}}(c) = \mathit{null}$ are set to *useless* when Algorithm 1 terminates.

In case (2) it can be shown that the update will set all $\Theta_{\bar{k}}(c)$ which are *null* to *optimal* after some time. For the same reasons as in the proof of Proposition 1, after some time all vertices are either closed or candidate. Let $v \in F_{\bar{k}}$ be such a vertex, with $\gamma_{\bar{k}}(v) = c$, and $\Theta \neq \mathit{null}$. If v is candidate it means that $\Theta_{\bar{k}}(c) = \mathit{optimal}$ because there is no open vertex, and thus, in particular, no open vertex v' such that $g_{\bar{k}}(v') + h_{\bar{k}}(v', c) < g_{\bar{k}}(v)$. If v is closed it means that v has been the candidate vertex with the smallest $R_{\bar{k}}$ at sometimes, and thus $\Theta_{\bar{k}}(c)$ as been set to *optimal*.

In all cases $\Theta_j(c) \neq \mathit{null}$, which proves the first part of Proposition 4.

The second part of the proposition is straightforward. Only initialization and final update can set $\Theta_{\bar{k}}(c)$ to *useless*. It is not possible that $\Theta_{\bar{k}}(c) = optimal$ before initialization. Moreover, final update can only change those $\Theta_{\bar{k}}(c)$ equal to *null*. Thus it is not possible for $\Theta_{\bar{k}}(c)$ to be set to *useless* after having being set to *optimal*. Only update can set $\Theta_{\bar{k}}(c)$ to *optimal*. At that time the only c such that $\Theta_{\bar{k}}(c) = useless$ come from initialization. It means that they are such that no goal vertex with that color exists in $\mathcal{P}_{\bar{k}}$. Thus, it is not possible that a vertex with that color becomes either open or candidate. Hence, it is not possible to change $\Theta_{\bar{k}}(c)$ from useless to optimal.

This ends the proof of the second part of Proposition 4. □

Proposition 5. *For any $c \in \Gamma$ if $\Theta_{\bar{k}}(c) = optimal$ then the value of $G_{\bar{k}}(c)$ is the optimal cost in $\mathcal{P}_j$ for reaching a goal vertex with color c. Moreover, if $\Theta_{\bar{k}}(c) = useless$, c can not be the color reached by a globally optimal solution.*

Proof. If $\Theta_{\bar{k}}(c) = optimal$ it means that, at some time, there existed $v \in F_{\bar{k}}$ such that v was open or candidate and $\gamma_{\bar{k}}(v) = c$, and there were no open vertex $v' \in V_{\bar{k}}$ such that $g_{\bar{k}}(v') + h_{\bar{k}}(v', c) < g_{\bar{k}}(v)$. At this time $G_{\bar{k}}(c)$ had been set equal to $g_{\bar{k}}(v'')$, where v'' is a goal vertex with color c minimizing the value of $g_{\bar{k}}$. $G_{\bar{k}}(c)$ is thus the cost of a path reaching color c in $\mathcal{P}_{\bar{k}}$. Assume it is possible to find a path with cost c_m strictly smaller than $G_{\bar{k}}(c)$. Let v''' be the goal vertex with color c reached by this path. By a similar argument than in the proof of the original A^* algorithm, either (1) $g_{\bar{k}}(v''') = c_m$, or (2) there exists an open vertex v'''' such that $g_{\bar{k}}(v'''') + h_{\bar{k}}(v'''', c) \leq c_m$. In case (1) v''' could have been selected as a goal vertex with color c minimizing the value of $g_{\bar{k}}$, this is in contradiction with the fact that $c_m < G_{\bar{k}}(c)$. In case (2) the existence of v'''' is in contradiction with the fact that there were no open vertex $v' \in V_{\bar{k}}$ such that $g_{\bar{k}}(v') + h_{\bar{k}}(v', c) < g_{\bar{k}}(v)$. This proves the first part of the Proposition 5.

If $\Theta_{\bar{k}}(c) = useless$ two cases are possible. Either no goal vertex exists with color c, in this case c can clearly not be the color reached by a globally optimal solution. Or Algorithm 1 stopped and $\Theta_{\bar{k}}(c)$ has been set to *useless* during final update. In this case it is possible that no global solution exists, so c can not be the color reached by a globally optimal solution. It is also possible that a global solution exists, reaching color c'. In this case, necessarily, $c' \neq c$. This is due to the fact that, in this case, Algorithm 1, outputs this solution, and thus, just before that, a candidate vertex v of color c' had the minimal value of $R_{\bar{k}}$, thus $\Theta_{\bar{k}}(c')$ has been set to *optimal*. As a globally optimal solution exists reaching $c' \neq c$ it is not possible that a globally optimal solution exists reaching c. Hence, if $\Theta_{\bar{k}}(c) = useless$, c can not be the color reached by a globally optimal solution. This proves the second part of Proposition 5. □

3.4 Running example

Consider the graph of Figure 2. Heuristics h_1 should have the following properties: $h_1(i_1, r) \leq 1$, $h_1(v_1, r) \leq 0$, and $h_1(v, b) \leq +\infty$ for any v. In the same way the values of H_2 (provided to φ_1 by φ_2) should always be such that: $H_2(r) \leq 2$, and $H_2(b) \leq 2 + 0 = 2$.

Assume φ_1 is running Algorithm 1 on $\mathcal{P}_1$. Initially, i_1 is open, $g_1(i_1) = 0$, and $R_1(i_1) = g_1(i_1) + \min_{c \in \{r,b\}}(h_1(i_1, c) + H_2(c))$. All other vertices are closed and such that g_1 is infinite. Moreover, $\Theta_1(b) = useless$, as no goal vertex with color b exists in $\mathcal{P}_1$. The first execution of the while loop will directly call the expand function, as i_1 is not candidate. It will be marked as closed (as i_1 is not a goal vertex). As $g_1(i_1) = 0 \leq 0 = g_1(i_1) + 0$, i_1 will not be re-opened. As $g_1(v_1) = +\infty > 1 = g_1(i_1) + 1$, v_1 will be opened, with $g_1(v_1) = 1$, and $R_1(v_1) = g_1(v_1) + \min_{c \in \{r,b\}}(h_1(v_1, c) + H_2(c))$, and $pred(v_1) = i_1$. After that, the expand function terminates. Immediately, $\Theta_1(r) = optimal$ as v_1 is a goal state with color r and no other open

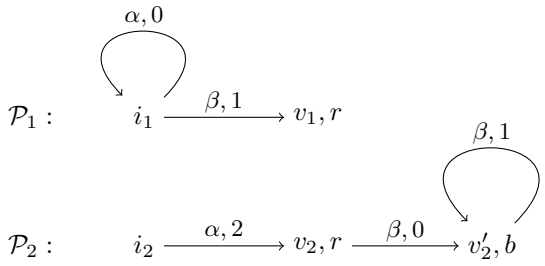


Figure 2: A CFS problem. Goal vertices are represented with their color (ex. v_1 is a goal with color r). Costs and labels are written above edges.

vertex exists, $G_1(r) = g_1(v_1) = 1$. As there is open vertices, a second execution of the while loop starts. The open or candidate vertex with minimal value of R_k is v_1. As v_1 is not candidate, a call to expand occurs immediately. As $v_1 \in F_1$, it is now candidate, and $R_1(v_1) = g_1(v_1) + G_2(r)$ if $\Theta_2(r) = \mathit{optimal}$ or $R_1(v_1) = g_1(v_1) + H_2(r)$ else. As v_1 has no neighbors, no new vertices are opened. From that, a new execution of the while loop occurs. As v_1 is candidate it is checked if it allows to conclude. No more calls to expand function occur as there no longer exists open vertices. As soon as $\Theta_2(r) = \mathit{optimal}$, with $G_2(r) = 2$ it is possible to conclude. The only possible local solution is to go from i_1 to v_1 in one step. Its cost is 1 locally, but $1 + 2 = 3$ globally, as the part of the solution in $\mathcal{P}_2$ is to go from i_2 to v_2 in one step.

4 Distributed planning with two components

This section extends the algorithm proposed to solve CFS problems to the more general framework of distributed planning (DP) problems, still in the limited case of two components. Compared to CFS, DP problems introduce two difficulties. First, colors are assigned dynamically to vertices: the color of vertex v is not given in advance by some coloring function γ, but is set as a function of the path p leading to this vertex $v = p^+$. Secondly, rather than a finite set Γ of colors, one potentially has an infinite set, since the idea is the the 'color' of vertex $v = p^+$ is the sequence of shared actions met along path p leading to v. And there is generally no (efficient) bound on the number of shared actions in a globally optimal distributed plan[2].

Let us recast a DP problem $(\mathcal{P}_1, \mathcal{P}_2)$ as a CFS problem $(\mathcal{P}'_1, \mathcal{P}'_2)$ with color set $\Gamma = (\Lambda_1 \cap \Lambda_2)^*$, the set of sequences of shared actions. One has $\mathcal{P}'_k = (V'_k, E'_k, \Lambda'_k, \lambda'_k, c'_k, i'_k, F'_k)$ with $V'_k = V_k \times \Gamma$, $E'_k = \{((v,w),(v',w')) \ : \ (v,v') \in E_k \wedge w' = w\, \pi_{\Lambda_1 \cap \Lambda_2}(\gamma_k((v,v')))\}$, $i'_k = (i_k, \varepsilon)$, $F'_k = F_k \times \Gamma$, $\gamma'_k : F'_k \to \Gamma$ is such that $\gamma'_k((v,w)) = w$, and c'_k is such that $c'_k(((v,w),(v',w'))) = c_k((v,v'))$. $(\mathcal{P}'_1, \mathcal{P}'_2)$ has however a major difference with CFS problems considered in Section 3: V'_1, E'_1, V'_2, and E'_2 may be infinite.

The remaining of this section is dedicated to extending the results of Section 3 to the case of the particular infinite graphs considered here. It will allow to use Algorithm 1 along with *expand* function given in Algorithm 3 for solving DP problems. This new *expand* function is in fact responsible for computing parts of P'_k from P_k (only when needed). Three points have to be addressed: (1) computation of $R_k((v,w))$ sometimes implies to take a minimum over an

[2]Strictly speaking, there exists a bound since the product planning problem $\mathcal{P} = \mathcal{P}_1 \times \mathcal{P}_2$ is finite, and no optimal plan will cross twice the same global state. However, the interest of distributed planning is to deploy local reasonings in $\mathcal{P}_k$ without making assumptions on the size of the 'external' component $\mathcal{P}_{\bar{k}}$.

infinite number of elements, (2) termination of the algorithm relies on finiteness of the graphs, (3) computation of $G_{\bar{k}}$ and $\Theta_{\bar{k}}$ are not directly possible on infinite graphs as non-accessible colors can not be determined at initialization.

Algorithm 3 expand function

```
{expand function has been called with an argument v of the form (v', w)}
if v' ∈ F_k then
  mark v = (v', w) candidate
  calculate R_k(v)
else
  mark v = (v', w) closed
end if
for all v'' such that (v', v'') ∈ E_k do
  w' ← wπ_Γ(γ_k((v', v'')))
  g_k((v'', w')) ← min(g_k((v'', w')), g_k((v', w)) + c_k((v', v'')))
  if g_k((v'', w')) strictly decreased then
    mark (v'', w') open
    pred((v'', w')) ← (v', w)
  end if
  calculate R_k((v'', w'))
end for
```

4.1 Computation of R_k and $H_{\bar{k}}$

For any color w (or at least any color which may correspond to an optimal distributed plan), $H_{\bar{k}}(w)$ should give a lower bound on the cost of reaching this color in $\mathcal{P}_{\bar{k}}$. Clearly, taking $H_{\bar{k}}(w) = 0$ for any w gives such a lower bound. However, it is usually better to get a tight bound in order to avoid as much exploration of the graphs as possible. For practical use of our algorithm, using a more accurate $H_{\bar{k}}$ would be recommended. An example of such an $H_{\bar{k}}$ is the following, where $w' < w$ is notation for w' is a prefix of w (recall that $H_{\bar{k}}$ is computed by $\varphi_{\bar{k}}$):

$$H_{\bar{k}}(w) = \min(H^o_{\bar{k}}(w), H^c_{\bar{k}}(w))$$

with:

$$H^o_{\bar{k}}(w) = \min_{\substack{(v_{\bar{k}},w') \text{ open} \\ w'<w}} (g_{\bar{k}}((v_{\bar{k}}, w')), h_{\bar{k}}((v_{\bar{k}}, w'))),$$

$$H^c_{\bar{k}}(w) = \min_{\substack{(v_{\bar{k}},w') \text{ candidate} \\ w'<w}} (g_{\bar{k}}((v_{\bar{k}}, w'))).$$

Notice that for any w it is possible to compute $H_{\bar{k}}(w)$, as the set of open and candidate (v, w) is always finite. Notice also that all $H_{\bar{k}}(w)$ can be computed by φ_k from a finite number of them given by $\varphi_{\bar{k}}$: the one which are such that $(v_{\bar{k}}, w)$ is open or candidate. We denote them by $\hat{H}_{\bar{k}}$. One then has: $H_{\bar{k}}(w) = \min_{w'<w} \hat{H}_{\bar{k}}(w')$.

When (v, w) is candidate, the computation of R_k is not an issue, it can be done exactly as in the simpler cases. Notice that, when (v, w) is candidate, v is necessarily in F_k. Hence, when $\Theta_{\bar{k}}(w) = \mathit{optimal}$ one has:

$$R_k((v, w)) = g_k((v, w)) + G_{\bar{k}}(w),$$

and in other cases:

$$R_k((v,w)) = g_k((v,w)) + H_{\bar{k}}(w).$$

However, when (v,w) is open it is not possible to directly use the previous definition of $R_k((v,w))$ as it may involve the computation of a minimum over an infinite number of elements. First of all, computing R_k as before would require the computation of $h_k((v,w),w')$ for any color w'. We consider instead $h_k((v,w)) = \min_{w'} h_k((v,w),w')$, which is computable with standard heuristic computation technics as a lower bound on the cost of a path in $\mathcal{P}_k$ from v to a goal vertex.

From that, when (v,w) is open, we suggest to compute $R_k((v,w))$ as follows:

$$R_k((v,w)) = g_k((v,w)) + h_k((v,w)) + \min_{w'>w} H_{\bar{k}}(w').$$

The second difficulty is that there may be an infinite number of colors w to consider when computing $\min_{w'} H_{\bar{k}}(w') = \min_{w'>w} H_{\bar{k}}(w')$. This suggest to add a constraint on $H_{\bar{k}}$: it should be such that $\min_{w'>w} H_{\bar{k}}(w')$ is computable for any w. Fortunately, using the implementation of $H_{\bar{k}}$ proposed above it is possible. One just has to remark that:

$$\min_{w'>w} H_{\bar{k}}(w') = \min(H_{\bar{k}}(w), \min_{w'>w} \hat{H}_{\bar{k}}(w')),$$

as the number of w such that $\hat{H}_{\bar{k}}(w)$ is defined is always finite, this minimum can be computed.

4.2 Termination of the algorithm

The main difference here with the case of CFS problems is that the termination of the algorithm is not ensured when there is no solution. This is due to the fact that the graph to explore is in general infinite. In fact, it is possible to ensure termination, as there is a bound on the length of the color corresponding to a possible solution. This bound can be computed by considering the number of vertices in the product of $\mathcal{P}_1$ and $\mathcal{P}_2$: if a solution exists, one is such that it passes at most one time in each vertex of this graph. However, it is not straightforward to tightly compute this bound in a distributed manner. For this reason, in the following we focus on the case where there exists a solution.

Theorem 2. *In this context, if the considered planning problem $(\mathcal{P}_1,\mathcal{P}_2)$ has a solution, then: any execution of Algorithm 1 by φ_k on $\mathcal{P}_k$ terminates. Moreover, the output of Algorithm 1 for agent φ_k is a goal vertex $v_k \in F_k$, reached by a local plan p_k. The assembling of p_1 and p_2 provided by agents φ_1 and φ_2 resp. yields an optimal distributed plan (p_1,p_2) solving $(\mathcal{P}_1,\mathcal{P}_2)$.*

To prove theorem 2 one first prove that as soon as a solution exists for $(\mathcal{P}_1,\mathcal{P}_2)$, Algorithm 1 terminates and outputs a vertex (Proposition 6). After that it is sufficient to notice that the proof of Proposition 3 never relies on the assumption that V_1, E_1, V_2, or E_2 are finite, so this proposition also applies here.

Proposition 6. *When $(\mathcal{P}_1,\mathcal{P}_2)$ has a solution, Algorithm 1 terminates by outputting some v when executed by φ_k on $\mathcal{P}_k$.*

Proof. Assume $(\mathcal{P}_1,\mathcal{P}_2)$ has a solution but Algorithm 1 does not terminate. It means there is always an open or candidate couple (v,w). After some time, such a couple will be candidate. This is due to the fact that there exists a reachable goal vertex (else no solution would exist). After some time, any candidate couple which may be part of a solution will become the couple

with the minimal value for R_i. This is due to the fact that (1) all couples (v', w') marked as open by a call to expend function with argument (v, w) are such that $g_k((v', w')) \geq g_k((v, w)) + c$, where c is the minimal cost of an edge in $\mathcal{P}_k$, (2) any candidate couple (v, w) which may be part of a solution is such that $R_k((v, w)) < +\infty$, and (3) as for any w, $\Theta_{\bar{k}}(w)$ has to take a value after some time and Algorithm 1 does not terminate, any candidate couple with minimal value for R_k will be discarded after some time. After some time a couple (v, w) such that $\Theta_{\bar{k}}(w) = \mathit{optimal}$ will become candidate. Once again this is due to the existence of a solution. From the previous argument, such a couple will become the couple with minimal value for R_k. Thus, Algorithm 1 will terminate thanks to this couple. This is in contradiction with the assumption taken, thus this assumption is false, which means that when $(\mathcal{P}_1, \mathcal{P}_2)$ has a solution, Algorithm 1 terminates.

Assume Algorithm 1 terminates without outputting a (v, w). It means that, at some time, all couples (v, w) were closed. However, as $(\mathcal{P}_1, \mathcal{P}_2)$ has a solution there exists a reachable couple (v', w') such that $v' \in F_k$ and after some time $\Theta_{\bar{k}}(w') = \mathit{optimal}$. As all couples (v, w) were closed it means that all the reachable part of $\mathcal{P}'_k$ has been explored (each (v, w) has been marked open at some time). As v' is a goal vertex, (v', w') has been marked candidate after being marked open. As after some time $\Theta_{\bar{k}}(w') = \mathit{optimal}$ it is not possible that (v', w') has been marked closed before Algorithm 1 terminated. This is in contradiction with our assumption, thus, Algorithm 1 can only terminate by outputting some (v, w). □

4.3 Computation of $G_{\bar{k}}$ and $\Theta_{\bar{k}}$

As before, these two functions have to be computed by agent $\varphi_{\bar{k}}$ independently of $\mathcal{P}_k$, and in particular, independently of G_k and Θ_k. A possible implementation, where $\Theta_{\bar{k}}$ and $G_{\bar{k}}$ are computed along execution of Algorithm 1 by $\varphi_{\bar{k}}$, is the following:

- initialization: $\forall w \in \Gamma$, $\Theta_{\bar{k}}(w)$ is considered as *null* and $G_{\bar{k}}(w) = +\infty$ (but only the $\Theta_{\bar{k}}(w) \neq \mathit{null}$ and the corresponding values of $G_{\bar{k}}$ are stored).
- update (1): as soon as there exists $v \in F_{\bar{k}}$ such that (v, w) is open or candidate, and there is no open couple (v', w') such that $g_{\bar{k}}((v', w')) + h_{\bar{k}}((v', w')) < g_{\bar{k}}((v, w))$ and $w' < w$, $\Theta_{\bar{k}}(\gamma_{\bar{k}}(v)) = \mathit{optimal}$ and
$$G_{\bar{k}}(w) = \min_{v' \in F_{\bar{k}}} g_{\bar{k}}((v', w)).$$
- update (2): as soon as for a given w there exists no $w' < w$ and v such that (v, w') is open or (v, w) is candidate, if $\Theta_{\bar{k}}(w) = \mathit{null}$, then $\Theta_{\bar{k}}(w)$ is set to *useless*.
- final update: when Algorithm 1 stops, for all $w \in \Gamma$ such that $\Theta_{\bar{k}}(w) = \mathit{null}$, set $\Theta_{\bar{k}}(w) = \mathit{useless}$, and $G_{\bar{k}}(w) = +\infty$.

Proposition 4 and Proposition 5 still hold with these new manners of computing $\Theta_{\bar{k}}$ and $G_{\bar{k}}$. The only difference in the proofs is about detection of non-reachable colors. One just has to remark that, after some time, any non-reachable color w will be such that no prefix w' of w appears in a couple (v, w') either open or candidate. Hence, after some time, $\Theta_{\bar{k}}(w)$ will be equal to *useless* thanks to update (2). After that it will never become equal to *optimal* as no prefix w' of w exists such that a couple (v, w') is either open or candidate, so update (1) will never occur for w.

4.4 Running example

Consider the graph of Figure 3.

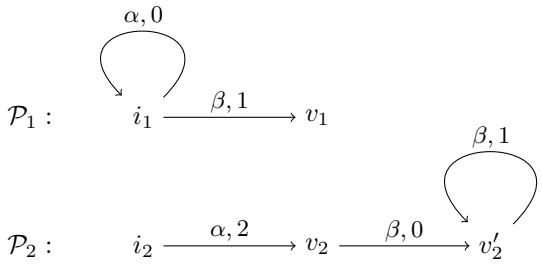


Figure 3: A DP problem. All non-initial vertices are goal. Costs and labels are written above edges.

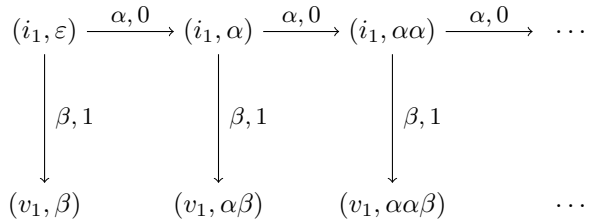


Figure 4: From CFS to DP.

Applying the transformation of DP problems into CFS problems proposed above , the graph $\mathcal{P}_1'$ would be as depicted in Figure 4.

An execution of Algorithm 1 by φ_1 on $\mathcal{P}_1$ starts with (i_1, ε) open. Then a call to expand function closes (i_1, ε) and opens (i_1, α) and (v_1, β). After that depending on the values of the different heuristics, a call to expand function will occur on either (i_1, α) or (v_1, β). Assume it is called on (i_1, α). Then (i_1, α) is closed and $(i_1, \alpha\alpha)$ and $(v_1, \alpha\beta)$ are opened. After that expand will be called on either (v_1, β), $(i_1, \alpha\alpha)$ or $(v_1, \alpha\beta)$. Which will either mark (v_1, β) or $(v_1, \alpha\beta)$ candidate, or close $(i_1, \alpha\alpha)$ and open $(i_1, \alpha\alpha\alpha)$ and $(v_1, \alpha\alpha\beta)$. After each time an element $(v_1, w\beta)$ is opened with $w \in \{\alpha\}^*$, $\Theta_1(w\beta) = \mathit{optimal}$ and $G_1(w\beta) = |w|.0 + 1$. As all costs of edges are positive, any open element of the form $(v_1, w\beta)$ with $w \in \{\alpha\}^*$ becomes candidate after a finite time. After some time $\Theta_2(\beta) = \mathit{useless}$ (it is not possible to reach a goal state in G_2 using only one edge with color β), and $\Theta_2(\alpha\beta) = \mathit{optimal}$ with $G_2(\alpha\beta) < \min(H_2(w\beta), G_2(w\beta))$ for all $w \in \{\alpha\}^*$ such that $\Theta_2(w\beta) \neq \mathit{optimal}$ and $G_2(\alpha\beta) < G_2(w\beta)$ for all $w \in \{\alpha\}^*$ such that $\Theta_2(w\beta) = \mathit{optimal}$. It allows φ_1 to conclude that its part of the optimal solution (which has a global cost of 3) reaches v_1 with color $\alpha\beta$. Moreover, the values of *pred* allow to conclude that the path in $\mathcal{P}_1$ should be to loop on i_1 one time and then go to v_1.

5 Conclusion

A* is a celebrated depth-first search algorithm to quickly find a shortest path in a possibly huge oriented graph. Its variants are extremely used to solve optimal planning problems, which are weak versions of optimal control problems. This paper has presented A#, a multi-agent version of A*, dedicated to quickly find an optimal multi-agent strategy to drive a distributed

system to a target state. Its convergence and its consistency have been proved. Compared to other approaches to factored/distributed planning, this is the first distributed search algorithm that also provides a globally optimal plan. Moreover, it is not hierarchical in the sense that all components run the same algorithm simultaneously, and bias each other's searches by their own guesses. It is thus more a 'consensus' approach than a master-slave approach.

The practical interest of A# will be soon tested on random planning benchmarks. In terms of theoretical complexity, however, the worst case bounds of factored planning problems remain unchanged, and one may have to explore the whole local graph of each local planning problem. However, compared to running A* on the equivalent product problem, one may be exponentially faster when components are loosely coupled. This is the standard key advantage of distributed planning, which explores possible plans as partial orders of actions rather than sequences, and thus saves the exploration of different interleavings of concurrent actions.

While the paper is limited to two components, A# extends to graphs of interacting components in a usual manner (these graphs are obtained by placing an edge between components that share actions). In particular, A# may become very efficient when the interaction graph between components is a tree.

Beyond intensive benchmarking of A#, our future work will examine its extension to the efficient distributed detection of problems that have no solution, which remains an open question.

References

[1] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100 –107, 1968.

[2] S. Edelkamp. Planning with pattern databases. In *Proceedings of the 6th european conference on planning*, pages 13–24, 2001.

[3] M. Helmert, P. Haslum, and J. Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In *In Proc. ICAPS 2007*, pages 176–183, 2007.

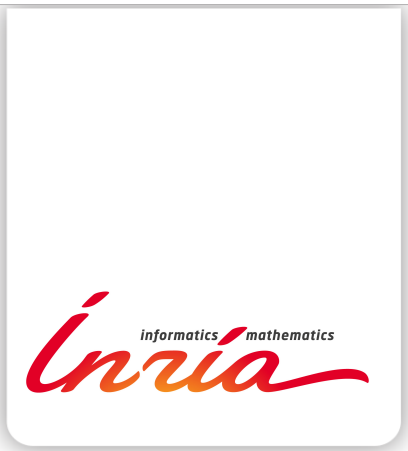
[4] E. Karpas and C. Domshlak. Cost-optimal planning with landmarks. In *Proceedings of the 21st international joint conference on Artifical intelligence*, pages 1728–1733, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[5] R. Brafman and C. Domshlak. Factored planning: How, when, and when not. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-2006)*, pages 809–814, 2006.

[6] R. Brafman and C. Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS*, pages 28–35, 2008.

[7] E. Amir and B. Engelhardt. Factored planning. In *In IJCAI,Äô03*, pages 929–935. Morgan Kaufmann, 2003.

[8] E. Fabre and L. Jezequel. Distributed optimal planning: an approach by weighted automata calculus. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 211 –216, dec. 2009.



RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399